3501MO0018

Autonomous Robots

Universitat de Girona

EKF Map based localization of a Mobile robot

Fernando Cervigni Martinelli
Andrew Durrant

Girona, 2nd April 2009

1. Introduction

In this report, the challenge of a robot 2D-self-localisation, with a priori knowledge of the map of operation, is investigated. Differently from the previous assignment, where the Bayes theorem was applied with a generic probability density function in order to find the current state belief, this work deals exclusively with single-gaussian-like *pdfs*. This simplifies considerably the calculations required to deal with the localisation task, since the whole state belief can be described by only a few parameters, namely the state vector mean and standard deviation.

The Kalman Filter has been proved to be the optimal solution for systems modelled by state variables that contain gaussian-white noise[1]. However, even assuming that the robot location probability density function is a Gaussian, the linear version of the Kalman Filter cannot be used to solve this problem, since it is inherently non-linear[2]. For that reason an extended version of the Kalman Filter is implemented. This version, called EKF, is very similar to the linear one except for the fact that it uses jacobians instead of linear matrices. These jacobians are calculated with the system being iteratively linearized around the updated state variables, which yields very good results despite the approximations being carried out.

2. Problem definition

This simplified 2D localisation problem is comprised of one mobile robot, whose localisation is intended to be calculated as precisely as possible, and tens of pillars, which are the a priori known features that will help the robot in its localisation task.

In order to be able to sense the pillars, the robot has been provided with a sensor that informs it the distance and the angle orientation of a given pillar with

[1] It is often incorrectly assumed that Gaussian noise is necessarily white noise, yet neither property implies the other. Gaussianity refers to the probability distribution with respect to the value i.e. the probability that the signal has a certain given value, while the term 'white' refers to the way the signal power is equally distributed in the power spectrum.

[2] The non-linearity comes from the polar-coordinate-like transformation between the world coordinate reference and the range and bearing read by the sensor. Further detail is given in section 3.

respect to its own position. To simulate a real world context, the sensor gives readings with error – which is known to be gaussian – and is prone to failure, returning empty values to the CPU of the robot. The latter has to be able to treat that errors in the best way possible.

Another sensorial information is the odometry readings. Again, simulating real world characteristics, this measurements contain noise.

The final objective in this work is then, to concatenate and balance these two pieces of information in an optimal manner, so that the uncertainty of the robot regarding its own pose is minimized.

3. Proposed solution – EKF

A good way of representing and illustrating complex designs is by using block diagrams. Figure 1 shows an overview of the whole algorithm used to tackle the problem at hand. It is based upon the theory of the extended Kalman Filter, which is explained in detail, and has its main parts, like the calculation of the Jacobians, derived step by step in the following sections.

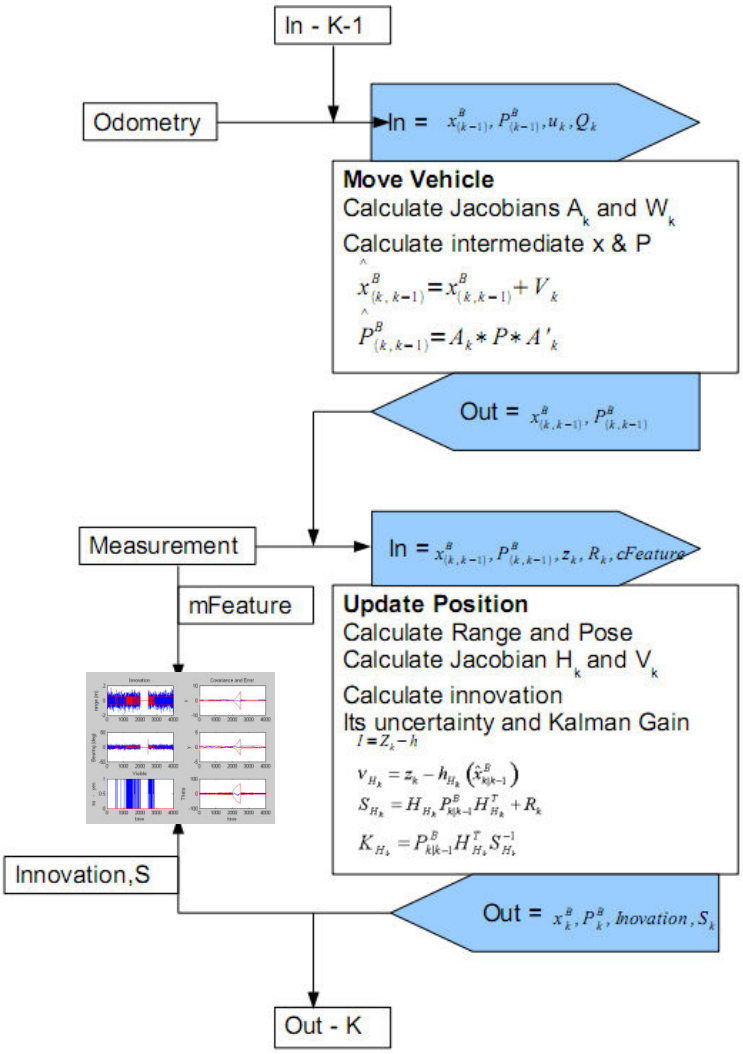

Figure 1 – Diagram of the proposed solution using EKF.

3.1: Function Move vehicle

The function *"move_vehicle"* is the function that, based on the odometry readings, updates the state variable –the position and orientation of the robot – and its uncertainty, represented here by the covariance matrix P. In the implemented Matlab code, the function which provides the odometry readings has been provided as a ready-to-use function.

As inputs, the function takes the current state and its uncertainty as well as the odometry and its uncertainty. The outputs are the current state and its uncertainty updated:

$$\left[x^B_{k|k-1}, P^B_{k|k-1}\right] = move\_vehicle\left(x^B_{k-1}, P^B_{k-1}, u^{R_{k-1}}_{R_k}, Q_k\right)$$

Eq. 1 and Eq. 2 show, respectively, how the outputs are calculated in function of the inputs.

$$\hat{x}^B_{k|k-1} = \hat{x}^B_{R_{k-1}} \oplus \hat{u}^{R_{k-1}}_{R_k} \quad \text{Eq. 1}$$

$$P^B_{k|k-1} = A_k P_{k-1} A^T_k + W_k Q_k W^T_k \quad \text{Eq. 2}$$

The operator $\oplus$ is defined as the compounded sum, which means that the addition being carried out is actually summing vectors of two different reference systems. This operation could be replaced by an ordinary addition except with the *u* vector being pre-multiplied by the rotation matrix $R^B_{R_{k-1}}$, which describes the rotation of the robot reference coordinate system with respect to the world's one.

In order to compute Eq. 1 and Eq. 2, the definition of the matrices A_k and W_k is necessary. These matrices are actually the jacobians of the *"move_vehicle"* function with respect, respectively, to the state variable x – Eq. 3 – and to its noise *w* – Eq. 4.

$$A_k = \left.\frac{\partial f(x_{k-1}, u_k, w_k)}{\partial x_{k-1}}\right|_{(\hat{x}_{k-1}, u_k, 0)} \quad \text{Eq. 3}$$

$$W_k = \left.\frac{\partial f(x_{k-1}, u_k, w_k)}{\partial w_k}\right|_{(\hat{x}_{k-1}, u_k, 0)} \quad \text{Eq. 4}$$

And from the literal expression of *f,*

$$f\left(\mathbf{x}^B_{R_{k-1}}, \mathbf{u}^{R_{k-1}}_{R_k}, \mathbf{w}_k\right) = \begin{pmatrix} \mathrm{c}\theta^B_{R_{k-1}}\left(x^{R_{k-1}}_{R_k} + w_{x_{jk}}\right) - \mathrm{s}\theta^B_{R_{k-1}}\left(y^{R_{k-1}}_{R_k} + w_{y_{jk}}\right) + x^B_{R_{k-1}} \\ \mathrm{s}\theta^B_{R_{k-1}}\left(x^{R_{k-1}}_{R_k} + w_{x_{jk}}\right) + \mathrm{c}\theta^B_{R_{k-1}}\left(y^{R_{k-1}}_{R_k} + w_{y_{jk}}\right) + y^B_{R_{k-1}} \\ \theta^B_{R_{k-1}} + \theta^{R_{k-1}}_{R_k} + w_{\theta_k} \end{pmatrix},$$

we can derive the Jacobians A_k and W_k as follows in Eq. 5 and Eq. 6:

$$A_k = \begin{pmatrix} 1 & 0 & -s\theta^B_{R_{k-1}} \cdot x^{R_{k-1}}_{R_k} - c\theta^B_{R_{k-1}} \cdot y^{R_{k-1}}_{R_k} \\ 0 & 1 & \mathrm{c}\theta^B_{R_{k-1}} \cdot x^{R_{k-1}}_{R_k} - \mathrm{s}\theta^B_{R_{k-1}} \cdot y^{R_{k-1}}_{R_k} \\ 0 & 0 & 1 \end{pmatrix} \quad \text{Eq. 5}$$

$$W_k = \begin{pmatrix} c\theta^B_{R_{k-1}} & -s\theta^B_{R_{k-1}} & 0 \\ s\theta^B_{R_{k-1}} & c\theta^B_{R_{k-1}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{Eq. 6}$$

This completely defines the function *"move_vehicle".* The implementation in Matlab code is as follows:

```
function [xHat_B_kIk_1, PHat_B_kIk_1] = move_vehicle(x_B_kIk_1, P_B_kIk_1,
uk, Qk)
    x_k_1 = x_B_kIk_1(1);
    y_k_1 = x_B_kIk_1(2);
    theta_k_1 = x_B_kIk_1(3);

    xR=uk(1);
    yR = uk(2);
    thetaR=uk(3);

    xHat_B_kIk_1= Compound(x_B_kIk_1, uk);

    %Jacobians
    Ak = [  1 0 -sin(theta_k_1)*xR-cos(theta_k_1)*yR;...
            0 1 cos(theta_k_1)*xR-sin(theta_k_1)*yR;...
            0 0 1];

    Wk = [  cos(theta_k_1) -sin(theta_k_1) 0;...
            sin(theta_k_1) cos(theta_k_1) 0;...
            0 0 1];

    PHat_B_kIk_1 = Ak*P_B_kIk_1*Ak' + Wk*Qk*Wk';
```

3.2: Function Update Position

The second part of the update of the state belief is based on the measurements of the sensor for a given pillar ahead of the robot. It is important to emphasize that in this problem, the data association problem has been omitted. Put another way, besides having the information of the distance and bearing of the pillar being measured, we are also provided with the information of exactly which pillar is being measured. In a real world situation, the pillars would most likely be very similar one to another, creating the problem of estimating which is the pillar being sensed.

As inputs, this function takes the current state and its uncertainty as well as the sensor reading – z_k – and its uncertainty – R_k. The outputs are again the current state and its uncertainty updated:

$$\left[x^B_k, P^B_k\right] = update\_position\left(x^B_{k|k-1}, P^B_{k|k-1}, z_k, R_k\right)$$

Eq. 7 and Eq. 8 show, respectively, how the outputs are calculated in function of the inputs.

$$\hat{x}_k^B = \hat{x}_{k|k-1}^B + K_{H_k} v_{H_k} \quad \text{Eq. 7}$$

$$P_k^B = \left(I - K_{H_k} H_{H_k}\right) P_{k|k-1}^B \left(I - K_{H_k} H_{H_k}\right)^T + K_{H_k} R_k K_{H_k}^T \quad \text{Eq. 8}$$

K_{H_k}, v_{H_k}, S_{H_k} are calculated as follows in Eq. 9, Eq. 10 and Eq. 11 respectively.

$$K_{H_k} = P_{k|k-1}^B H_{H_k}^T S_{H_k}^{-1} \quad \text{Eq. 9}$$

$$v_{H_k} = z_k - h_{H_k}\left(\hat{x}_{k|k-1}^B\right) \quad \text{Eq. 10}$$

$$S_{H_k} = H_{H_k} P_{k|k-1}^B H_{H_k}^T + R_k \quad \text{Eq. 11}$$

Before the continuation of the derivation of functions H_{H_k} and h_{H_k} in the next paragraphs, an interesting analysis of the real signification of these equations can be performed. It is known that the updated position of the robot will be somewhere between the position it believes it is at, represented by *x*, and the position it should be so that the readings of the sensor would be consistent to its knowledge of the map.

Let us analyze the two extremes of this problem. First, assume that the current location of the robot is uncertain but the readings of the sensor are perfect, that is, have no errors. In this case, the matrix R_k would be zero, resulting Eq. 9 in $K_{H_k} = {H_{H_k}}^{-1}$. Substituting this K_{H_k} in Eq. 7 and Eq. 8 provides a zero covariance matrix *P* – which makes total sense since the precise information of the sensor perfectly defined the position of the robot – and a new *x* such that the discrepancy v_{H_k} is zero.

Now assume that the sensor is imprecise but the current position is perfectly known, that is, *R* is different from zero but not *P.* This assumption leads $K_{H_k} = 0$ which results in not changing the state variable *x* nor the covariance matrix *P* at all. This makes sense since if we know our position without error and also know that the sensor information has error, the obvious decision is to completely ignore it.

Closing this parenthesis, let us return to the derivation of function *"Update Position"*. Analogue to the function *"move_vehicle",* in order to compute Eq. 7 ,Eq. 8,

Eq. 9, Eq. 10 and Eq. 11, the matrix H_{H_k}, as well as the function h_{H_k} have to be defined.

The function h_{H_k} yields the readings of distance – from now on referred to as ρ – and bearing – from now on referred to as β – of the robot with respect to the pillar in question – see Eq. 12 below.

$$h(\mathbf{x}, \text{pillar}) = h\left(\begin{pmatrix} x_k \\ y_k \\ \theta_k \end{pmatrix}, \begin{pmatrix} x_f \\ y_f \end{pmatrix}\right) = \begin{pmatrix} \rho \\ \beta \end{pmatrix} \quad \text{Eq. 12}$$

The pillar is fully described by its *x* and *y* coordinates, here referred to as x_f and y_f. The calculation of those two variables is described and further detailed with the help of the support diagram in Figure 2 below.

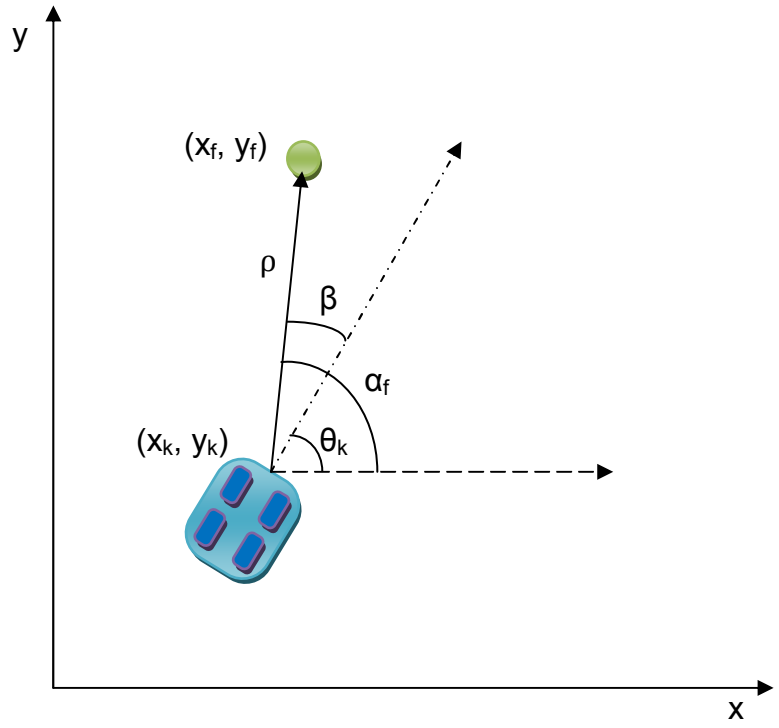

Figure 2 – Diagram of the calculation of sensor bearing and range.

From Figure 2 it becomes clear that:

$$\beta = \alpha_f - \theta_k \quad \text{Eq. 13}$$

And we know also that:

$$\alpha_f = \text{atan2}\left(\frac{y_f - y_k}{x_f - x_k}\right) \quad \text{Eq. 14}$$

Hence, substituting Eq. 14 in Eq. 13 we have the second component of the function *h,* considering already the addition of the Gaussian white noise in the bearing measurement v_β:

$$h_2 = \beta = \operatorname{atan2}\left(\frac{y_f - y_k}{x_f - x_k}\right) - \theta_k + v_\beta \qquad \text{Eq. 15}$$

As for the first component of *h*, the simple Pythagoras relationship can be drawn:

$$h_1 = \rho = \sqrt{\left(y_f - y_k\right)^2 + \left(x_f - x_k\right)^2} + v_\rho \qquad \text{Eq. 16}$$

Where v_ρ is the Gaussian white noise affecting the distance measurement.

Having the function *h* fully defined, we are able to derive its Jacobian, *H*, with respect to the state variable *x*. The result is as follows in Eq. 17:

$$H = \begin{pmatrix} \frac{-x_0}{\rho} & \frac{-y_0}{\rho} & 0 \\ \frac{1}{1+\frac{{y_0}^2}{{x_0}^2}} \cdot \frac{y_0}{{x_0}^2} & \frac{-1}{\left(1+\frac{{y_0}^2}{{x_0}^2}\right).\frac{1}{x_0}} & -1 \end{pmatrix} \qquad \text{Eq. 17}$$

With the notation simplifications: $x_0 = x_f - x_k$ and $y_0 = y_f - y_k$.

The function “Update_Position” is implemented in Matlab as follows:

```
function [x_B_k, P_B_k, Innovk, S] = update_position(xHat_B_kIk_1,
PHat_B_kIk_1, zk, Rk, cFeature)

    xf = cFeature(1);
    yf = cFeature(2);
    xk = xHat_B_kIk_1(1);
    yk = xHat_B_kIk_1(2);
    th_k = xHat_B_kIk_1(3);
    range = sqrt((xf-xk)^2+(yf-yk)^2);
    slope = (yf-yk)/(xf-xk);

    % evaluate the non-linear function at xHat, with vk=0
    h = [range;Angle_mod_360(atan2(yf-yk,xf-xk)-th_k)];

    % Jacobians
    Hk = [-(xf-xk)/range -(yf-yk)/range 0;
         1/(1+slope^2)*(yf-yk)/(xf-xk)^2, -1/(1+slope^2)/(xf-xk) -1];
    Vk = [1 0;0 1];

    % Innovation and its uncertainty
    Innovk=zk-h;
    Innovk(2) = Angle_mod_360(Innovk(2));

    S=Hk*PHat_B_kIk_1*Hk'+Rk;
```

```
% Kalman gain
Kk = PHat_B_kIk_1*Hk'*S^-1;
x_B_k = xHat_B_kIk_1 + Kk*Innovk;
P_B_k = (eye(3,3)-Kk*Hk)*PHat_B_kIk_1*(eye(3,3)-Kk*Hk)'+Kk*Rk*Kk';
```

The support function *"AngleWrap"* ensures that the result of any operation involving angles throughout the program is kept between –π and +π. For example, if this function had not been implemented, the logical verification of whether the absolute value of the angle of a new feature seen by the robot is in its field of view would not work.

4. Simulation Results

The program has been implemented in steps, the first step being the implementation and simulation of the robot without any sensorial information, that is, only counting on its odometry to locate itself.

As expected and discussed in the lectures, the error accumulates indefinitely, because there is no absolute information to curb the uncertainty increase. The growing ellipses in Figure 3, representing the covariance of the state variable, illustrate how the error grows differently in the directions *x* and *y*.

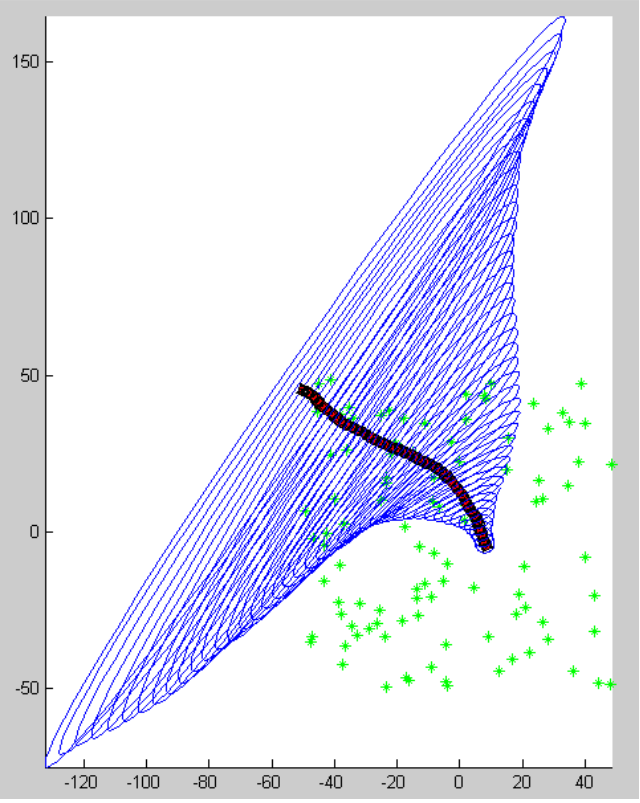

Figure 3 – Simulation without sensor information. Error grows indefinitely.

When the sensor measurement feedback is implemented, an enormous improvement is noticed, since the error is kept bounded by the sensor readings. These provide, although with error, absolute localisation information thanks to the exact knowledge of the scene map.

Figure 4 shows how the robot keeps its localisation estimation error bounded when considering the sensor readings. However, from the iteration *'Faultingk0'* until the iteration *'Faultingk0'* + *'Faultgap'*[3], a sensor failure is simulated, which causes the ellipses to start growing, just like in Figure 3. When the sensor regains its functionality, though, the error falls sharply and stays at a low level until the end of the simulation.

It is interesting to notice that, even though the robot starts with zero angular velocity, the noise added to the state variable, which includes the heading of the robot as its third component, makes the robot change randomly its direction, leading it to eventually turn.

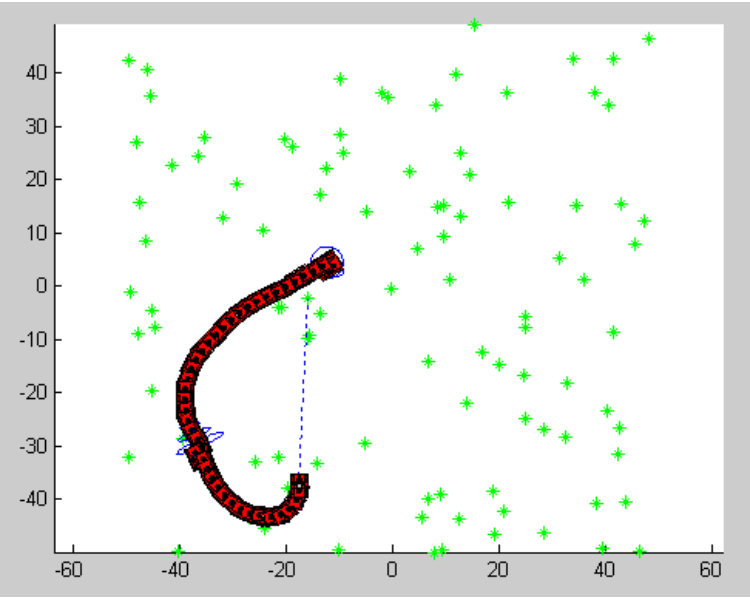

Figure 4 – Simulation with sensor information. Error is kept bounded.

The final statistics of the simulation are shown in Figure 5 below. We can notice the unbounded error growing for some instants when, after the middle of the

[3] For the program simulated the parameter *'Faultingk0'* was set to 2000 and *'Faultgap'* to 500.

simulation, the sensor wrongly returned empty values to the *"Update Position"* function.

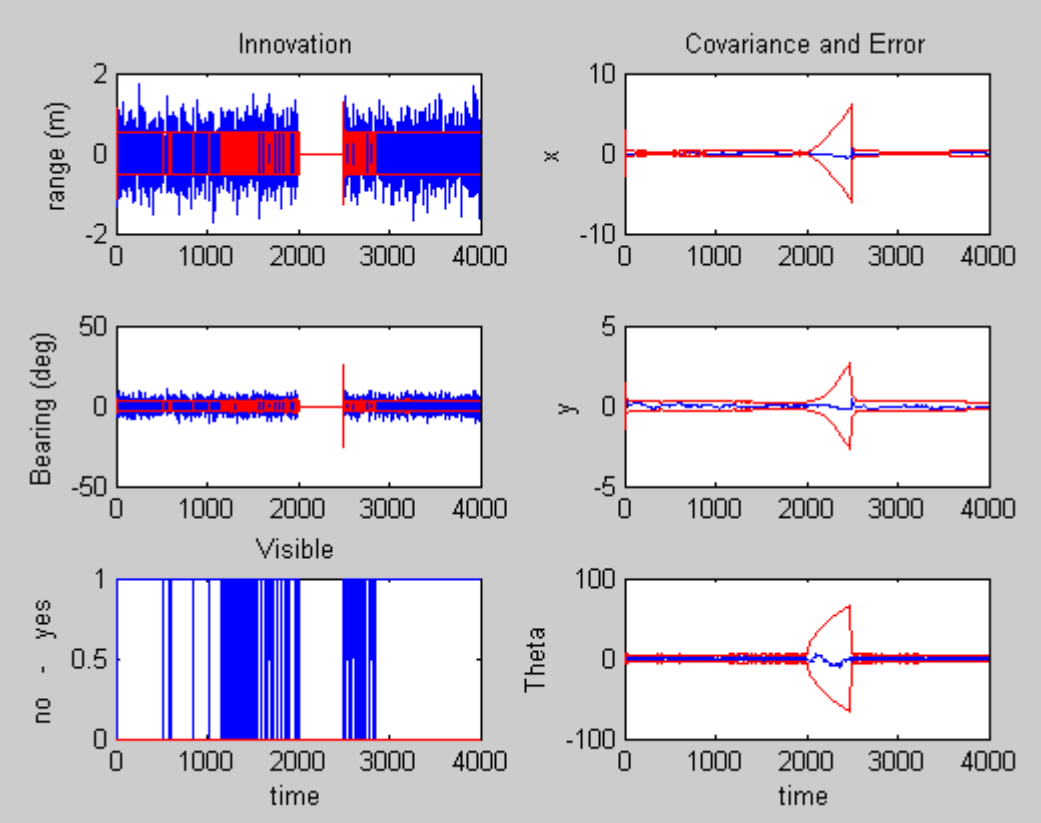

Figure 5 – Simulation statistics.

The innovation, as defined in Eq. 10, shows the errors throughout the simulation both in range, top left graph, and also in bearing, mid-left graph. These errors are the discrepancies between the actual sensor readings and the expected ones according to the robot belief of the world map – which is absolute – and its belief of its own location – which has imprecisions.

5. Conclusion

The aims of this section were all achieved with little unexpected errors that were all corrected relatively easily. The results were as expected, the simulations of the robot operating with no references to features has an ever-growing uncertainty. That uncertainty is related to the distance travelled with only odometry as a reference. Once the references to the map features are added then a bounded error in displacement could be attained.

The ready-to-modify code, with a well designed animation really helped the students to see in practice the behaviour of each attempt of modelling the measurement and motion histograms. Hence, debugging and improving the implementation of the code created was made much easier than in other laboratory activities of the masters. Put another way, the students efforts were focused on what really mattered, not on operational things as drawing graphics or developing simulations. The pre-prepared exercises are an example to be followed by the other subjects.

Implementing and deriving the three Jacobians for the calculations was a difficult task to perform because the calculation of matrices of partial derivative is prone to simplistic human error. The resultant equations have to be extensively cross checked. Still, unexplained errors often lead back to incorrect derivations at this stage.

This project has been highly motivating for its practicality and for its challenging aspect. The knowledge acquired throughout this activity will surely be very important for all the following subjects involving the motion of autonomous robots. It became also clear that dead-reckoning localisation systems alone are not viable, because, regardless of how good and fine they are, the statistical error they provide is intrinsically unlimited.

A fruitful extension to this problem could be the solution of the data association problem where the feature whose information is captured is unknown and must be estimated.